# R基础

**1. 背景知识**

**R起源**  
R 是一个有着统计分析功能及强大作图功能的软件系统，是由奥克兰大学统计学系的Ross Ihaka和Robert Gentleman 共同创立。由于R 受Becker, Chambers & Wilks 创立的S 和Sussman 的Scheme两种语言的影响，所以R 看起来和S 语言非常相似。

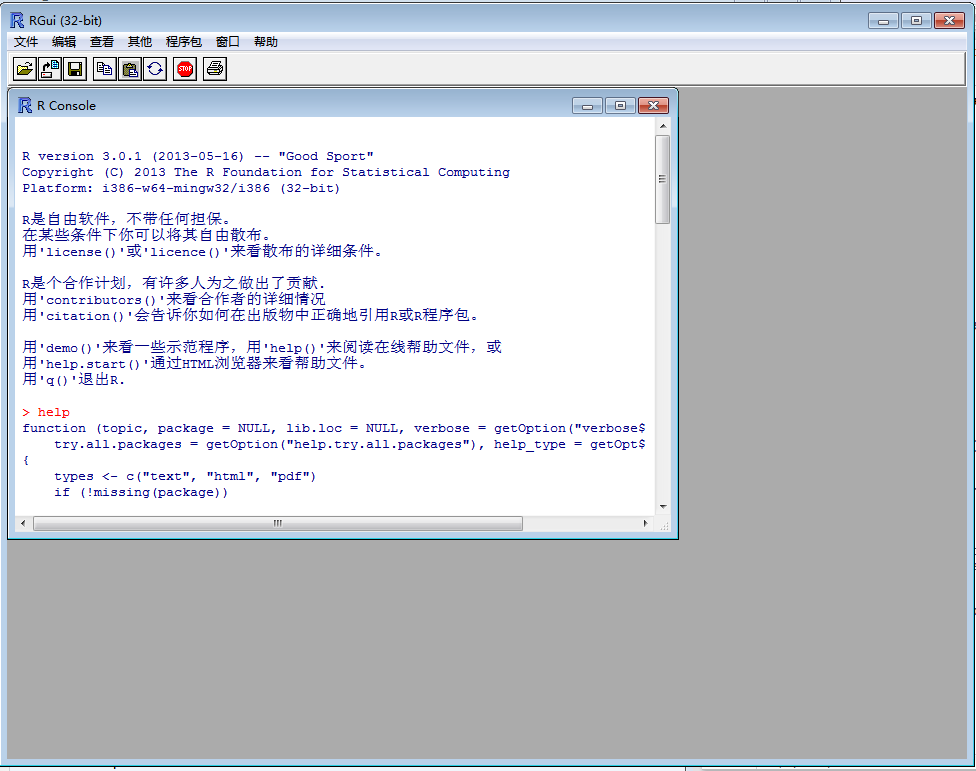
R 是一个世界范围统计工作者共同协作的产物，至2013 年2 月共计近5000 个包可在互联网上自由下载，这些都是各行业数据分析同行的工作结晶。

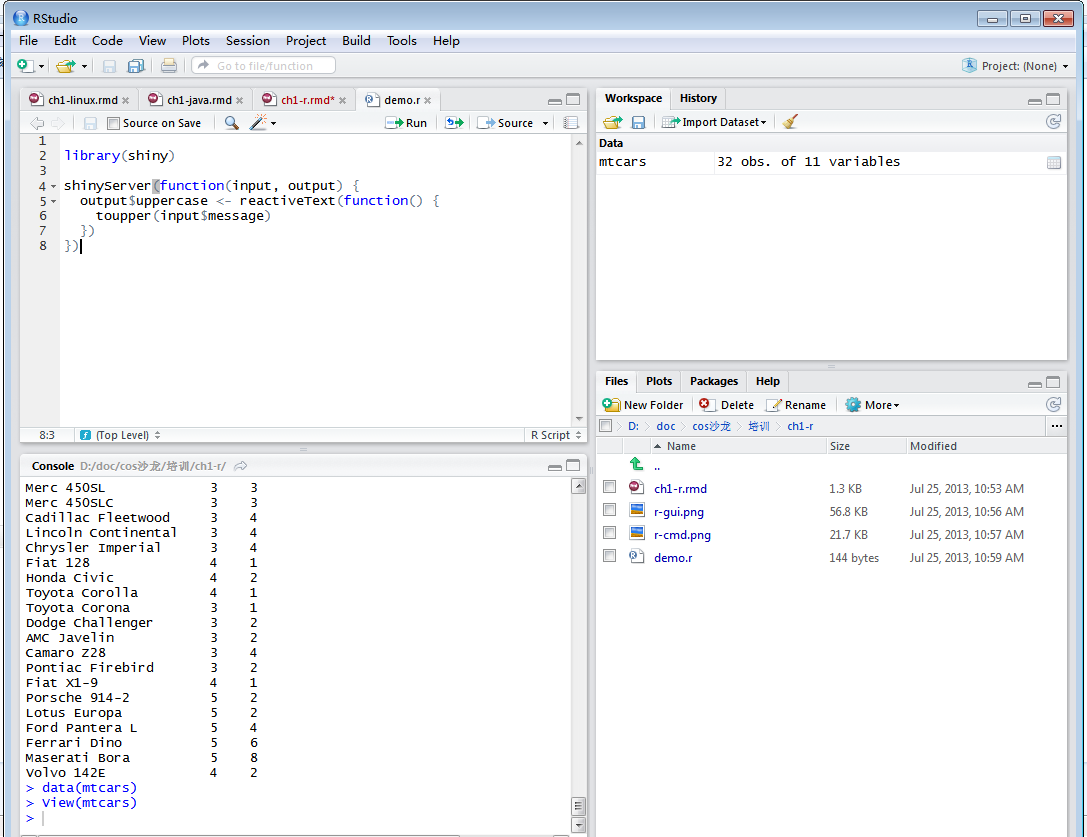
**R的特点**

1. 有效的数据处理和保存机制。
2. 拥有一整套数组和矩阵的操作运算符。
3. 一系列连贯而又完整的数据分析中间工具。
4. 图形统计可以对数据直接进行分析和显示，可用于多种图形设备。
5. 一种相当完善、简洁和高效的程序设计语言。它包括条件语句、循环语句、用户自定义的递归函数以及输入输出接口。
6. R语言是彻底面向对象的统计编程语言。
7. R语言和其它编程语言、数据库之间有很好的接口。
8. R语言是自由软件，可以放心大胆地使用，但其功能却不比任何其它同类软件差。
9. R语言具有丰富的网上资源

**R的下载和安装**  
R是一个免费的自由软件，它有UNIX、LINUX、MacOS和WINDOWS版本，都是可以免费下载和使用的,在那儿可以下载到R的安装程序、各种外挂程序和文档。在R的安装程序中只包含了8个基础模块，其他外在模块可以通过CRAN获得。

**2. 开发环境**

R默认的界面环境：  
[](http://blog.fens.me/wp-content/uploads/2013/07/r-gui.png)

RStudio的IDE开发环境： http://www.rstudio.com/  
[](http://blog.fens.me/wp-content/uploads/2013/07/r-studio.png)

**3. R语法**

R是一种语法非常简单的表达式语言(expression language),**大小写敏感。**  
可以在R 环境下使用的命名字符集依赖于R 所运行的系统和国家(系统的locale 设置),允许数字,字母,“.”,“\_”

**1). 命名**  
命名必须以”.”或者字母开头，以”.”开头时第二个字符不允许是数字。

**2). 基本命令**  
基本命令要么是表达式（expressions），要么就是赋值（assignments）。

* 表达式，命令将被解析，并将结果显示在屏幕上，同时清空该命令所占内存。
* 赋值，命令将被解,并把值传给变量,但结果不会自动显示在屏幕上。

命令可以被”;”隔开或者另起一行。基本命令可以通过大括弧{},放在一起构成一个复合表达式。

注释：一行中以井号”#”开头  
换行：如果一条命令在一行结束的时候在语法上还不完整，换行提示符，默认是+

**3). 基本的对象**  
R创建和控制的实体被称为对象。它们可以是变量，数组，字符串，函数，或者其他通过这些实体定义的一般性的结构。

* 矩阵（matrix）或者更为一般的数组（array）是多维的广义向量。实际上，它们就是向量，而且可以同时被两个或者更多个索引引用，并且以特有的方式显示出来。
* 因子（factor）为处理分类数据提供的一种有效方法。
* 列表（list）是一种泛化（general form）的向量。它没有要求所有元素是同一类型，许多时候它本身就是向量和列表类型。列表为统计计算的结果返回提供了一种便利的方法。
* 数据框（data frame）是和矩阵类似的一种结构。在数据框中，列可以是不同的对象。可以把数据框看作是一个行表示观测个体并且（可能）同时拥有数值变量和分类变量的`数据矩阵’ 。许多实验数据都可以很好的用数据框描述：处理方式是分类变量而响应值是数值变量。
* 函数（function）是可以保存在项目工作空间的R 对象。该对象为R 提供了一个简单而又便利的功能扩充方法。见编写你自己的函数

在R会话过程中，对象是通过名字创建和保存的。objects(), ls()可以显示当前会话的对象名字。rm()可以删除对象。

**对象持久化**  
R 会话中创建的所有对象可以永久地保存在一个文件中以便于以后的R 会话调用。在每一次R 会话结束的时候，你可以保存当前所有可用的对象。如果你想这样做，这些对象将会写入当前目录下一个叫.RData的文件中，并且所有在这次会话中用过的命令行都会被保存在.Rhistory 的文件中。当R 再次在同一目录下启动，这些对象将从这个文件中重新导入工作空间。同时，相关的历史命令文件也会被导入。

**4). 向量和赋值**  
向量是由一串有序数值构成的序列

x <- c(10.4, 5.6, 3.1, 6.4, 21.7)

函数c()完成的赋值语句。这里的函数c() 可以有任意多个参数，而它返回的值则是一个把这些参数首尾相连形成的向量。

赋值也可以用函数assign()实现。

assign("x", c(10.4, 5.6, 3.1, 6.4, 21.7))

赋值符<-,->可以看作是该命令一个语义上的缩写。

c(10.4, 5.6, 3.1, 6.4, 21.7) -> x

**向量运算**  
在算术表达式中使用向量将会对该向量的每一个元素都进行同样算术运算。  
出现在同一个表达式中的向量最好是长度一致。如果他们的长度不一样，该表达式的值将是一个和其中最长向量等长的向量。  
表达式中短的向量会被循环使用以达到最长向量的长度。  
对于一个常数就是简单的重复。

v <- 2\*x + y + 1

**逻辑向量**  
逻辑向量元素可以被赋予的值,有TRUE，FALSE 和NA 逻辑向量可以由条件式（conditions）产生 temp <- x > 13

**字符向量**  
字符向量就是字符串，可以用双引号和单引号作分割符。

paste():可以把单独的字符连成字符串，可以有任意多的参数。参数中的任何数字都将被显式地强制转换成字符串，而且以同样的方式在终端显示。默认的分隔符是单个的空格符。

修改分隔符换成”“

labs <- paste(c("X","Y"), 1:10, sep="")

**索引向量：通过索引值可以选择和修改一个数据集的子集**  
一个向量的子集元素可以通过向量名后面的方括号中加入索引向量得到。如果一个表达式的结果是向量，则我们可以直接在表达式的末尾方括号中加入索引向量以得到结果向量的子向量.

* **逻辑向量**:索引向量必须和被挑选元素的向量长度一致。向量中对应索引向量元素为TRUE 的元素将会被选中，而那些对应FALSE 的元素则被忽略。

y <- x[!is.na(x)]

* **正整数向量**:索引向量必须是1, 2, … , length(x)的子向量。索引向量中索引对应的元素将会被选中，并且在结果向量中的次序和索引向量中的次序一致。这种索引向量可以是任意长度的，结果向量的长度和索引向量完全一致。

x[1:10]

* **负整数向量**:这种索引向量指定被排除的元素而不是包括进来。

y <- x[-(1:5)]

* **字符串向量**:这可能仅仅用于一个对象可以用names 属性来识别它的元素。名字向量的子向量可以像上面第二条提到的正整数标签一样使用。
* fruit <- c(5, 10, 1, 20)
* names(fruit) <- c("orange”, “banana”, “apple”, “peach”)
* lunch <- fruit[c(“apple”,“orange”)]

**5).运算符**  
算术运算符：

+ - \* /

逻辑运算符：

<，<=，>，>=，==,!=, &, |, !

数学函数：

log，exp，sin，cos，tan，sqrt ，max ，min，range，length，sum，prod，var

注：var(x): 等价于sum((x-mean(x))^2)/(length(x)-1)

**6). 控制语句**  
**条件语句:if**

if (expr1 ) expr2 else expr3

**循环控制：for,repeat,while**

for (name in expr1 ) expr2

其中name 是循环变量，expr1是一个向量表达式，而expr2常常是根据虚拟变量name 而设计的成组表达式。在name 访问expr1所有可以取到的值时，expr2都会运行。

**警告：相比其他程序语言，R代码里面很少使用for(),执行效率很低**

repeat expr

while (condition) expr

关键字break：可以用于结束任何循环，甚至是非常规的。它是结束repeat 循环的唯一办法。  
关键字next：可以用来结束一次特定的循环，然后直接跳入"下一次"循环。

**7). 生成正则序列**  
1:30 语句：等价于向量c(1, 2, …, 29, 30) 30:1 语句：可用来产生一个逆向的数列。  
seq：数列生成中最为常用的工具。seq(1,30,1) rep：把一个数的完整拷贝多次，保持数列顺序req(x,times=5)

**8). 缺损值**  
在某些情况下，向量的元素可能有残缺. 当一个元素或者值在统计的时候"不可得到"（not available）或者"值丢失" （missing value），相关位置可能会被保留并且赋予一个特定的值NA。 任何含有NA 数据的运算结果都将是NA。 函数is.na(x)返回一个和x同等长度的向量。它的某个元素值为TRUE 当且仅当x中对应元素是NA。

z <- c(1:3,NA); ind <- is.na(z)

第二种"缺损"值，也称为非数值NaN(Not a Number)

0/0 或 Inf

**9). 对象**  
**内在属性：模式和长度**  
R操作的实体是对象。实数或复数向量，逻辑向量和字符串向量之类的对象属于"原子"型的对象，因为它们的元素都是一样的类型或模式。R的对象类型包括数值型，复数型，逻辑型,字符型,和原生型。向量必须保证它的所有元素是一样的模式。因此任何给定的向量必须明确属于逻辑性，数值型，复数型，字符型或者原生型.

列表是任何模式的对象的有序序列。列表被认为是一种"递归"结构而不是原子结构,因为它们的元素可以以它们各自的方式单独列出。函数和表达式也是递归结构。

所有对象都有模式(mode)和长度(length)两个内在属性

mode(x),length(x)

**外部属性**  
attributes(x):给出对象当前定义的非内在属性的列表。  
attr(object, name): 可以用来设置的外部属性。

z<-c(1:3,NA)

attr(z, "name") <- "abc"

attributes(z)

**对象的类属性**  
R里面的所有对象都属于类(class)，可以通过函数class(x)查看。  
对于简单的向量，类是对应的模式(mode)："numeric"，"logical"，"character" 或者"list"  
其他的类型，像"matrix"，"array"，"factor" 和"data.frame" 就可能是其他值。

**10). 因子(factor)**  
假定我们有一份来自澳大利亚所有州和行政区的信息样本 以及他们各自所在地的州名。

state <- c("tas", "sa", "qld", "nsw", "nsw", "nt")

在字符向量中，"有序"意味着以字母排序的。

创建因子factor：

statef <- factor(state)

statef

[1] tas sa qld nsw nsw nt

Levels: nsw nt qld sa tas

levels():可以用来得到因子的水平（levels）。

levels(statef)

[1] "nsw" "nt" "qld" "sa" "tas"

**函数tapply()和不规则数组**  
为计算样本中每个州的平均收入，我们可以用函数tapply()：

incomes <- c(60, 49, 40, 61, 64, 60)

incmeans <- tapply(incomes, statef, mean)

> incmeans

nsw nt qld sa tas

62.5 60.0 40.0 49.0 60.0

函数tapply() 可以用来处理一个由多个分类因子决定的向量下标组合。例如，我们可能期望通过州名和性别把这税务会计师分类。

**有序因子**  
因子的水平是以字母顺序排列的，或者显式地在factor中指定。有时候因子的水平有自己的自然顺序并且这种顺序是有意义的。ordered()就是用来创建这种有序因子, ordered()和factor 基本完全一样。

大多数情况下，有序和无序因子的唯一差别在于前者显示的时候反应了各水平的顺序。在线性模型拟合的时候，两种因子对应的对照矩阵的意义是完全不同的。

**11). 数组**  
数组可以看作是带有多个下标类型相同的元素集合。  
维度向量(dimension vector)是一个正整数向量。如果它的长度为k，那么该数组就是k-维的。

向量只有在定义了dim属性后才能作为数组在R中使用。  
假定，z是一个含1500个元素的向量

z<-seq(1,1500)

dim(z) <- c(3,5,100)

attributes(z)

$dim

[1] 3 5 100

对dim属性的赋值使得该向量成一个3 ×5 ×100 的数组

**数组索引**  
数组元素可以通过给定数组名及其后方括号中用逗号隔开的下标访问。可以根据索引数组去给数组中不规则的元素集合赋值或者将数组中特定的元素返回到一个向量中

**array()**  
除了用设定一个向量dim 属性的方法来构建数组，它还可直接通过函数array将向量转换得到.

假定向量h 有24个或更少的数值，那么命令

h<-seq(1,24)

Z <- array(h, dim=c(3,4,2))

#等价操作

dim(Z) <- c(3,4,2)

**向量和数组混合运算**

* + 表达式运算是从左到右进行的
  + 短的向量操作数将会被循环使用以达到其他操作数的长度
  + 有且只有短的向量和数组在一起，数组必须有一样的属性dim，否则返回一个错误
  + 向量操作数比矩阵或者数组操作数长时会引起错误
  + 如果数组结构给定，同时也没有关于向量的错误信息和强制转换操作，结果将是一个和它的数组操作数属性dim 一致的数组。

**数组的外积**  
数组一个非常重要的运算就是外积运算（outer product）。如果a 和b 是两个数值数组，它们的外积将是这样的一个数组：维度向量通过连接两个操作数的维度向量得到；数据向量则由a的数据向量元素和b的数据向量元素的所有可能乘积得到。外积是通过特别的操作符%o%实现：

ab <- a %o% b

ab <- outer(a, b, "\*")

[,1] [,2] [,3]

[1,] 1 2 3

[2,] 2 4 6

[3,] 3 6 9

命令中的乘法操作符可以被任意一个双变量函数代替。

x<-c(1,2,3);y<-c(2,3,4)

f <- function(x, y) cos(y)/(1 + x^2)

z <- outer(x, y, f)

两个常规向量的外积是一个双下标的数组(就是矩阵，最大秩为1)。

**数组的广义转置**  
函数aperm(a, perm) 可以用来重排一个数组a

B <- aperm(z, c(2,1))

#等价操作

t(z)

**12). 矩阵**

矩阵是一个双下标的数组. R包括许多只对矩阵操作的操作符和函数。例如上面提到的t(X)就是矩阵的转置函数。函数nrow(A) 和ncol(A) 将会分别返回矩阵A 的行数和列数。

**矩阵相乘**  
操作符%\*% 用于矩阵相乘。  
如果A和B是大小一样的方阵，那么

A \* B

将是一个对应元素乘积的矩阵，而

A %\*% B

则是一个矩阵积。如果x 是一个向量，那么

x %\*% A %\*% x

crossprod():可以完成"矢积"运算，也就是说crossprod(X,y) 和t(X) %\*% y 等价，而且在运算上更为高效。  
diag():返回以该向量元素为对角元素的对角矩阵。

**性方程和求逆**  
求解线性方程组是矩阵乘法的逆运算。当下面的命令运行后，

b <- A %\*% x

如果仅仅给出A 和b，那么x 就是该线性方程组的根。在R 里面，用命令

solve(A,b)

矩阵的逆可以用下面的命令计算，

solve(A)

**特征值和特征向量**  
ev<-eigen(Sm):用来计算矩阵Sm 的特征值和特征向量。这个函数的返回值是一个 含有values 和vectors 两个分量的列表。

ev <- eigen(Sm)

ev$values表示Sm 的特征值向量，ev$vec 则是相应特征向量构成的一个矩阵。

**奇异值分解和行列式**  
svd(M): 可以把任意一个矩阵M作为一个参数, 且对M 进行奇异值分解。这包括一个和M 列空间一致的正交列U 的矩阵，一个和M 行空间一致的正交列V 的矩阵，以及一个正元素D 的对角矩阵，如M = U %\*% D %\*% t(V)。D 实际上以对角元素向量的形式返回。svd(M) 的结果是由d, u 和v构成的一个列表。

如果M 是一个方阵，就不难看出

absdetM <- prod(svd(M)$d)

计算M 行列式的绝对值。如果在各种矩阵中都需要这种运算，我们可以把它定义为一个R 函数

absdet <- function(M) prod(svd(M)$d)

此后, 我们可以把absdet() 当一个R 函数使用了。R有一个计算行列式的内置函数det和另外一个给出符号和模的函数。

evals <- eigen(Sm, only.values = TRUE)$values

**cbind()和rbind()构建分块矩阵**  
函数cbind() 和rbind():把向量和矩阵拼成一个新的矩阵。cbind() 把矩阵横向合并成一个大矩阵（列方式），而rbind()是纵向合并（行方式）。  
对数组实现连接操作的函数c()  
将一个数组强制转换成简单向量的标准方法是用函数as.vector()。

vec <- as.vector(X)

#等价操作

vec <- c(X)

**因子的频率表**  
单个因子会把各部分数据分成不同的组。类似的是，一对因子可以实现交叉分组等。函数table() 可以从等长的不同因子中计算出频率表。如果有k 个因子参数，那么结果将是一个k-维的频率分布数组。

statefr <- table(statef)

statefr <- tapply(statef, statef, length)

**13). 列表(list)**  
R的列表是一个以对象的有序集合构成的对象。列表中包含的对象又称为它的分量（components）。分量可以是不同的模式或类型，如一个列表可以同时包括数值向量，逻辑向量，矩阵，复向量，字符数组，函数等等。

Lst <- list(name="Fred", wife="Mary", no.children=3,child.ages=c(4,7,9))

分量常常会被编号，并且可以利用这种编号来访问分量。如果列表Lst 有四个分量，这些分量则可以用Lst[[1]], Lst[[2]], Lst[[3]] 和Lst[[4]] 独立访问。因为Lst 是一个列表，所以函数length(Lst) 给出的仅仅是分量的数目. 列表的分量可以被命名，这种情况下可以通过名字访问。

**构建和修改列表**  
list():将已有的对象构建成列表。  
Lst[5] <- list(matrix=Mat)

**列表连接**  
当连接函数c() 的参数中有列表对象时，结果就是一个列表模式的对象。它的分量是那些当作参数的列表。

list.ABC <- c(list.A, list.B, list.C)

**14). 数据框**  
数据框是一个属于data.frame类的列表。

对于可能属于数据框的列表对象有下面一些限制条件，

* + 分量必须是向量(数值, 字符, 逻辑)，因子，数值矩阵，列表或者其他数据框;
  + 矩阵，列表和数据框为新的数据框提供了尽可能多的变量，因为它们各自拥有列，元素或者变量;
  + 数值向量，逻辑值，因子保持原有格式，而字符向量会被强制转换成因子并且它的水平就是向量中出现的独立值;
  + 在数据框中以变量形式出现的向量结构必须长度一致，矩阵结构必须有一样的行数.
  + 数据框常常会被看作是一个由不同模式和属性的列构成的矩阵。

**创建数据框**  
可以通过函数data.frame 创建符合上面对列(分量)限制的数据框对象：

accountants <- data.frame(home=statef, loot=incomes)

符合数据框限制的列表可被函数as.data.frame() 强制转换成数据框。

**绑定任意的列表**  
attach() 是一个泛型函数。它不仅允许搜索路径绑定目录和数据框，而且还可以绑定其他对象。所以任何其他"list" 模式的对象都可以这样绑定：

attach(any.old.list)

任何被绑定的东西都可利用detach 通过位置编号或者名字（最好采用名字）去 除绑定。

**管理搜索路径**  
search(): 显示当前的搜索路径。它可以用来跟踪已被绑定或者绑定去除的列表和数据框(以及包)。

search()

[1] “.GlobalEnv” “Autoloads” “package:base”

其中.GlobalEnv 是工作空间

lentils 被绑定后，我们可以看到

search()

[1] ".GlobalEnv" "lentils" "Autoloads" "package:base"

ls(2)

[1] "u" "v" "w"

ls (或者objects) 可以用来查看搜索路径中任何位置的内容。

**15). 读数据**  
大的数据对象常常是从外部文件中读入，而不是在R 对话时用键盘输入的。  
**read.table()函数**  
为了可以直接读取整个数据框，外部文件常常要求有特定的格式。 第一行可以有该数据框各个变量的名字。 随后的行中第一个条目是行标签，其他条目是各个变量的值。  
**scan() 函数**  
假定有三个数据向量，长度一致并且要求并行读入。其中，第一个向量是字符模式，另外两个是数值模式，文件是input.dat。第一步是用scan() 以列表的形式读入这三个向量，  
**访问内置数据**  
R 提供了大约100个内置的数据集(在包datasets 中)，其他的包(包括和R捆绑发布的推荐包) 也提供了一些作为例子的数据集。可以用下面的命令查看当前能访问的数据集列表

data()

**从其他R 包里面导入数据**  
为了访问某个特定包的数据，可以使用参数package，例如

data(package=“rpart”)

data(Puromycin, package=“datasets”)

如果一个包已经被library 绑定，那么它的数据集会被自动地加入到R 的搜索路径中去。

**编辑数据**  
edit(x):调用数据框和矩阵时，R 会产生一个电子表形式的编辑环境。

xnew <- edit(xold)

**16) 编写函数**  
R语言允许用户创建自己的函数（function）对象,如mean(), var(),postscript() 等等,这些函数都是用R 写的，因此在本质上和用户写的没有差别。

一个函数是通过下面的语句形式定义的，

name <- function(args, ...) {}

其中expression 是一个R 表达式(常常是一个成组表达式)，它利用参数argi 计算最终 的结果。该表达式的值就是函数的返回值。 可以在任何地方以name(expr1 , expr2 , …) 的形式调用函数。

函数定义如下：

twosam <- function(y1, y2) {

n1 <- length(y1); n2 <- length(y2)

yb1 <- mean(y1); yb2 <- mean(y2)

s1 <- var(y1); s2 <- var(y2)

s <- ((n1-1)\*s1 + (n2-1)\*s2)/(n1+n2-2)

tst <- (yb1 - yb2)/sqrt(s\*(1/n1 + 1/n2))

tst

}

a<-1:3;b<-5:7

twosam(a,b)

**参数命名和默认值**  
和产生正则序列一样，如果被调用函数的参数以"name=object"方式给出，它们可以用任何顺序设置。但是，参数赋值序列可能以未命名的，位置特异性的方式给出，同时也有可能在这些位置特异性的参数后加上命名参数赋值。 因此，如果有下面方式定义的函数fun1

fun1 <- function(data, data.frame, graph, limit) {}

**… 参数**  
一个函数的参数设置可以传递给另外一个函数。这个可以通过给函数增加一个额外的参数来实现。  
举例如plot

plot

function (x, y, ...)

**在函数中赋值**  
注意任何在函数内部的普通赋值都是局部的暂时的，当退出函数时都会丢失。因此函数中的赋值语句X <- qr(X) 不会影响调用该函数的程序赋值情况。

“强赋值"操作符 <<- :如果想在一个函数里面全局赋值或者永久赋值

fscope<-function(){

a<-1

b<<-2

c=3

}

**作用域**  
函数内部的变量可以分为三类：形式参数(formal parameters)，局部变量(local variables),自由变量(free variables)。

* + 形式参数是出现在函数的参数列表中的变量。它们的值由实际的函数参数绑定形式参数的过程决定的。
  + 局部变量由函数内部表达式的值决定的。既不是形式参数又不是局部变量的变量是自由变量。
  + 自由变量如果被赋值将会变成局部变量

z<-45

f <- function(x) {

y <- 2\*x

print(x)

print(y)

print(z)

}

x 是形式参数，y 是局部变量，z 是自由变量。

**定制环境**  
可以修改位置初始化文件，并且每个目录都可以有它特有的一个初始化文件。利用函数.First 和.Last。位置初始化文件的路径可以通过环境变量R PROFILE 设置。这个文件包括你每次执行R时一些自动运行的命令。

**类，泛型函数和面向对象**  
一个对象的类决定了它会如何被一个泛型函数处理。相反，一个泛型函数由参数自身类的种类来决定完成特定工作或者事务的。如果参数缺乏任何类属性，或者在该问题中有一个不能被任何泛型函数处理的类，泛型函数会有一种默认的处理方式。

下面的一个例子使这个问题变得清晰。类机制为用户提供了为特定问题设计和编写泛型函数的便利。在众多泛型函数中，plot() 用于图形化显示对象，summary()用于各种类型的概述分析，以及anova() 用于比较统计模型。 能以特定方式处理类的泛型函数的数目非常庞大。

methods() 得到当前对某个类对象可用的泛型函数列表：

methods(class="data.frame")

相反，一个泛型函数可以处理的类同样很多。例如，plot() 有默认的方法和变 量处理对象类"data.frame"，"density"，"factor"，等等。

一个完整的列表同样可以通过函数methods():

methods(plot)

**17) R中的统计模型**  
线性模型,对于常规的多重模型拟合，最基本的函数是lm()。

fm2 <- lm(y ~ x1 + x2, data = production)

将会拟合y 对x1 和x2 的多重回归模型和一个隐式的截距项

**提取模型信息的泛型函数**  
lm() 的返回值是一个模型拟合结果对象；技术上就是属于类"lm” 的一个结果列表。关于拟合模型的信息可以用适合对象类"lm" 的泛型函数显示，提取，图示等等。

add1 coef effects kappa predict residuals alias

deviance family labels print step anova drop1

formula plot proj summary

* + anova(object1 , object2) 比较一个子模型和外部模型，并且产生方差分析表。
  + coef(object) 提取回归系数(矩阵)。全称：coefficients(object).
  + deviance(object) 残差平方和，若有权重可加权。
  + formula(object) 提取模型公式信息。
  + plot(object) 产生四个图，显式残差，拟合值和一些诊断图。
  + predict(object, newdata=data:frame) 提供的数据框必须有同原始变量一样标签的变量。结果是对应于data:frame中决定变量预测值的向量或矩阵。
  + predict.gam(object,newdata=data:frame) predict.gam() 是安全模式的predict()。它可以用于lm, glm和gam 拟合对象。在正交多项式作为原始的基本函数并且增加新数据意味着必须使用不同的原始基本函数。
  + print(object) 简要打印一个对象的内容
  + residuals(object) 提取残差(矩阵)，有权重时可加权，省略方式：resid(object)。
  + step(object) 通过增加或者减少模型中的项并且保留层次来选择合适的模型。在逐步搜索过程中，AIC (Akaike信息规范)值最大的模型将会被返回。
  + summary(object) 显示较详细的模型拟合结果

**18). 图形工具**  
图形工具是R 环境里面一个非常重要和多用途的组成部分。我们可以用这些图形工具显示各种各样的统计图并且创建一些全新的图。

图形工具既可交互式使用，也可以批处理使用。在许多情况下，交互式使用是最有效的。打开R 时，它会启动一个图形设备驱动（device driver）。该驱动会打开特定的图形窗口（graphics window）以显示交互式的图片。尽管这些都是自动实现的，了解用于UNIX 系统的X11() 命令和Windows 系统的windows() 命令是非常有用的。一旦设备驱动启动，R 绘图命令可以用来产生统计图或者设计全新的图形显示。

绘图命令可以分成了三个基本的类：

* + 高级绘图命令: 在图形设备上产生一个新的图区，它可能包括坐标轴，标签，标题等等。
  + 低级画图命令: 会在一个已经存在的图上加上更多的图形元素，如额外的点，线和标签。
  + 交互式图形命令: 允许你交互式地用定点设备（如鼠标）在一个已经存在的图上添加图形信息或者提取图形信息。

**高级绘图命令**  
plot(),这是一个泛型函数：产生的图形依赖于第一个参数的类型或者类。 pairs(X),描绘多元数据提供了两个非常有用的函数

**低级图形函数**  
高级图形函数不能准确产生你想要的图。低级图形命令可以在当前图上精确增加一些额外信息(如点，线或者文字)。 points(x, y) lines(x, y)

**数学标注**  
在某些情况下，在一个图上加上数学符号和公式是非常有用的。在R 里面，这可以通过函数expression 实现，

text(x, y, expression(paste(bgroup("(", atop(n, x), ")"), p^x, q^{n-x})))

**图像设备输出**

* + X11() 用UNIX 类型的系统的X11 桌面系统
  + windows() 用于Windows 系统
  + quartz() 用于MacOS X 系统
  + postscript() 用于PostScript 打印机，或者创建PostScript 文件。
  + pdf() 创建可以插入PDF 文件中PDF 文件
  + png() 创建PNG 位图文件。(不总是有效的：参考它的帮助文件)
  + jpeg() 创建JPEG 位图文件，非常适用于影

**19). 包(packages)**  
所有的R 函数和数据集是保存在包里面的。只有当一个包被载入时，它的内容才可以被访问。这样做一是为了高效，二是为了帮助包的开发者防止命名和其他代码中的名字冲突。

library():查看系统中安装的包 library(plyr):加载plyr包 CRAN.packages() 连接因特网，并且允许自动更新和安装包。 search()为了查看当前有那些包载入了

**标准包**  
标准包构成R 原代码的一个重要部分。它们包括允许R 工作的的基本函数，和本文档中描述的数据集，标准统计和图形工具。在任何R 的安装版本中，它们都会被自动获得。

**捐献包和CRAN**  
世界各地的作者为R 捐献了好几百个包。其中一些包实现了特定的统计方法，另外一些给予数据和硬件的访问接口，其他则作为教科书的补充材料。 可以从CRAN (http://CRAN.R-project.org/ 和它的镜像)和其他一些资源，如Bioconductor (http://www.bioconductor.org/) 下载得到

**命名空间**  
包有命名空间(namespaces),并且现在所有基本的和推荐的的包都依赖于包datasets。

它们允许包的作者隐藏函数和数据，即只允许内部用户使用，它们防止函数在一个用户使用相同名字时被破坏，它们提供了一种访问特定包的某个对象的方法。

有两个操作符和命名空间相关。 双冒号操作符:: 选择一个特定命名空间得到的函数定义。可以通过base::t 使用，因为它是在包base 中定义的。 三冒号操作符::: 可能会出现在一些R 代码中：它有点像双冒号操作符，但可以访问隐藏对象。

包常常是包之间依赖的（inter-dependent），载入其中一个可能会引起其他包的自动载入。

**4. R的扩展包**

**1). plyr (数据处理)**  
plyr是一个数据处理的包，可以把大的数据集通过一些条件拆分成小的数据集的功能包。

数据集baseball(http://www.baseball-databank.org/)  
21,699 records covering 1,288 players, spanning 1871 to 2007

举例：击球得分：RBI(Run batted in)

install.packages("plyr")

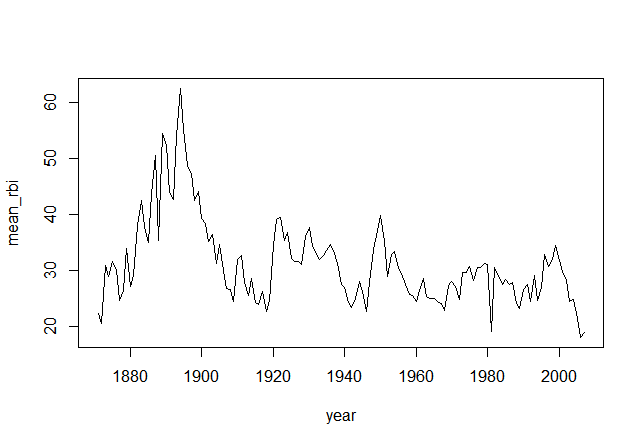
library(plyr)

?baseball

ddply(baseball, .(lg), c("nrow", "ncol"))

rbi <- ddply(baseball, .(year), summarise, mean\_rbi = mean(rbi, na.rm = TRUE))

plot(mean\_rbi ~ year, type = "l", data = rbi)

[](http://blog.fens.me/wp-content/uploads/2013/07/rbi-plyr.png)

**2). stringr (字符串处理)**  
stringr是一个字符串处理的包，可以方便地进行各种字符串的操作。

install.packages("stringr")

library(stringr)

fruits <- c("one apple", "two pears", "three bananas")

str\_replace(fruits, "[aeiou]", "-")

str\_replace\_all(fruits, "[aeiou]", "-")

str\_replace\_all(fruits, "([aeiou])", "")

str\_replace\_all(fruits, "([aeiou])", "\\1\\1")

str\_replace\_all(fruits, "[aeiou]", c("1", "2", "3"))

str\_replace\_all(fruits, c("a", "e", "i"), "-")

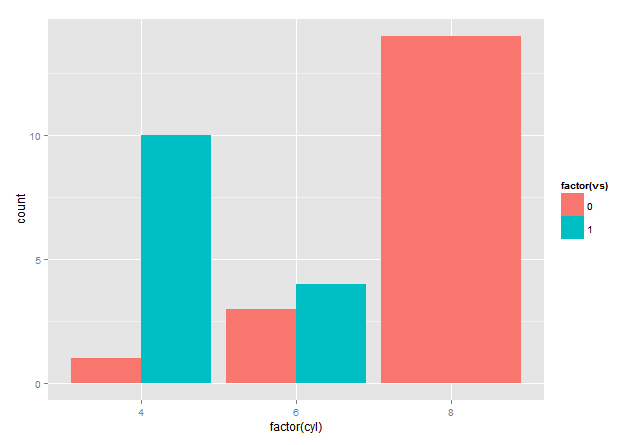
**3). ggplot2 (可视化)**  
ggplot2专业级的可视化绘图包

install.packages("ggplot2")

library(ggplot2)

g<-ggplot(mtcars, aes(x=factor(cyl), fill=factor(vs)))

g+geom\_bar(position="dodge")

[](http://blog.fens.me/wp-content/uploads/2013/07/ggplot2.png)